UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/816,103	03/31/2004	Edward T. Grochowski	42P18305	9899

59796 7590 12/29/2010

INTEL CORPORATION
c/o CPA Global
P.O. BOX 52050
MINNEAPOLIS, MN 55402

EXAMINER	
FENNEMA, ROBERT E	

ART UNIT	PAPER NUMBER
2183	

NOTIFICATION DATE	DELIVERY MODE
12/29/2010	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

heather.l.adamson@intel.com

Office Action Summary	Application No.	Applicant(s)	
	10/816,103	GROCHOWSKI ET AL.	
	Examiner	Art Unit	
	Robert Fennema	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 October 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,2,4-6,10-17,20-28,32-51 and 55-71* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-2, 4-6, 10-17, 20-28, 32-51, 55-71* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *10/14/2010*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-2, 4-6, 10-17, 20-28, 32-51, 55-71 have been considered. Claims 1, 17, 35-51, and 55 amended as per Applicant's request.

2. Examiner notes the entry of the following papers:

- Amendment to the claims and remarks filed 10/14/2010

- IDS filed 10/14/2010

Claim Objections

3. In Claim 17, Line 6, "to" should be inserted between "accessible" and "shreds".

4. In Claim 35, Line 14, "art" should read "are".

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. Claims 67 and 70-71 are rejected under 35 U.S.C. 102(e) as being anticipated by Kissell (USPN 7,376,954).

7. As per Claim 67, Kissell teaches: An apparatus, comprising:

a processor capable of user-level multithreading including (Column 4, Line 5):

a first group of resources to hold a per shared resource thread ("shred") application state for a first shred (Column 9, Lines 43-45) to be created by a first user-level instruction (Column 10, Line 37);

a second group of resources, which is to include a replicate of the first group of resources, to hold a per shred application state for a second shred (Column 9, Lines 43-45, they all contain their own versions of the same registers) to be created by a second user-level instruction (Column 10, Line 37); and

a third group of resources to be shared by the first shred and the second shred to hold a shared application state (Column 16, Lines 37-53, the inter-thread communications storage), wherein at least a portion of the third group of resources is to be directly updateable by the first and the second shred (Column 16, Lines 41-44, it can be updated by any thread using loads and stores) for communication between the first shred and the second shred (Column 16, Lines 37-54); and

wherein the first, second, and third group of resources are private from another privileged-level software entity created thread (Column 9, Lines 41-43, each process has its own address space, and is private from all other processes); and

execution resources to concurrently execute the first shred and the second shred (Column 4, Line 5).

8. As per Claim 70, Kissell teaches: The apparatus of Claim 67, wherein the shared application state is to be shared by a privileged-level software entity thread and is not to be shared with other privileged-level software entity threads, and wherein the privileged level software entity is an operating system (OS) (Column 9, Lines 41-43, the address space of a process is separate from other processes).

9. As per Claim 71, Kissell teaches: The apparatus of Claim 67, wherein the second group of resources is a copy of the first group of resources, and wherein the first group of resources includes a combination of resources selected from a group consisting of general registers, floating point registers, SSE registers, instruction pointer, and flags (Column 9, Lines 43-45), and wherein the third group of resources includes a combination of resources selected from a group consisting of general registers, floating point registers, shared communication registers, flags, memory management registers, address translation tables, privilege levels, and control registers (Column 9, Lines 41-43 and Column 13, Line 44).

Claim Rejections - 35 USC § 103

10. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

11. Claims 1-2, 4-6, 10-12, 15-17, 20, 22-25, 27-28, 32, 35-43, 47, 50-51, 55-58, and 62-65 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell, in view of Hennessy et al. ("Computer Architecture: A Quantitative Approach", herein Hennessy), further in view of Marshall ("Threads: Basic Theory and Libraries", herein Marshall).

12. As per Claim 1, Kissell teaches: A method, comprising:

executing a first Operating System (OS) generated thread generated by an OS, the OS generated thread being associated with a first system state (Column 8, Line 30, the Software Process A);

encountering a non-privileged, user-level programming instruction (Column 10, Line 37, FORK);

creating in hardware without intervention of the OS, responsive to the non-privileged level programming instruction, a first shared resource thread (shred) (Column 10, Lines 37-45, FORK creates a new thread, and Column 8, Lines 40-42 show that the OS is not used) that is associated with a first private portion of a first application state (Column 9, Lines 43-45, each thread has a separate state portion), wherein the first shred shares a shared portion of the first application state with at least a second shred (Column 9, Lines 41-43, they share the same address space);

executing, responsive to the non-privileged, user-level programming instruction, the first shred concurrently with at least one of the one or more other shreds (Column 2, Line 36), but fails to teach:

communicating, between the first shred and the second shred, via one or more shared registers that are adapted to be updateable by the first and the second shred, wherein the one or more shared registers are private to shreds that share the shared portion of the first application state and the first system state.

Kissell teaches a shared memory area updateable by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), and further teaches that every process has memory which is private to the process, but shared among all threads in the process (Column 9, lines 41-45, the shared memory space), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with less storage available), and the system can work with either type, as memory is memory.

Furthermore, as discussed above, Kissell discloses that every process has memory shared between all threads. While Kissell does not elaborate on how this shared memory is used, Marshall, a reference which discloses well known concepts of threads and processes, discloses that the shared memory space of processes are used

to share data between threads in a process (Page 2, where Marshall indicates that sharing data between threads is immediately available, while there is significant overhead in sharing data between processes). Therefore, one of ordinary skill in the art would have recognized, given how threads and processes work, that the shared memory space of the process, which is private to threads in the process, can be used to communicate data between threads, and as taught by Hennessy, could be implemented as registers if one wished to have higher speed communication at higher cost. Additionally, Examiner notes that in Kissell's description of the inter-thread communication storage, Kissell indicates that the storage "can" be made global, which implies that it is not necessarily inherently global. The inter-thread communication storage could easily be read as a description of the shared memory of a process which is made available to all threads of the system, as it describes portions of memory which threads can read or write from in order to pass data along to each other. Therefore, even without the well known teachings of Marshall and the private shared memory spaces in processes, Kissell could be interpreted as having the private storages for threads to communicate.

13. As per Claim 2, Kissell teaches: The method of claim 1, wherein the first private portion of the first application state is associated with at least one of a plurality of registers including general purpose registers, floating point registers, MMX registers, segment registers, a flags register, an instruction pointer, control and status registers, SSE registers, and a MXCSR register (It is inherent that each thread of execution has

these registers in their private state, or they cannot function as intended. Also see Column 9, Lines 43-45).

14. As per Claim 4, Kissell teaches: The method of claim 3, further comprising:

creating a second operating system generated thread to execute a second process in response to a privileged non-user level programming instruction, the second operating system generated thread to be associated with a second application state (Column 8, Line 30, Software Process C); and

creating a third shred, in response to encountering a second non-privileged user-level programming instruction associated with the second process, wherein the third shred is to be associated with a shared portion and a private portion of the second application state (Column 9, Lines 41-45),

wherein the second operating system generated thread and the third shred do not share a portion of the first application state (Column 9, Lines 41-45, different VPE's and processes do not share address space).

15. As per Claim 5, Kissell teaches: The method of claim 1, wherein the first shred and the second shred share a current privilege level and share a common address translation (Column 9, Lines 41-43, threads in the same process share a common address space and can share the same privilege level).

16. As per Claim 6, Kissell teaches: The method of claim 1, further comprising:

receiving a non-privileged user-level programming instruction that encodes a shred destroy operation (Column 10, Line 52, YIELD).

17. As per Claim 10, Kissell teaches: The method of claim 1, further comprising:

scheduling, responsive to a user-level programming instruction, the first shred and the second shred without intervention of the operating system (Column 5, Lines 29-40).

18. As per Claim 11, Kissell teaches: The method of claim 1, wherein:

said communicating is performed via a user-level shred signaling instruction (Column 12, Line 51, MTTR).

19. As per Claim 12, Kissell teaches: The method of claim 11, further comprising: storing at least the first private portion of the first application state in a memory responsive to receipt of a context switch request (inherent in a context switch).

20. As per Claim 15, Kissell teaches: The method of claim 1, wherein:
the shred is to perform input/output (I/O) operations (Column 5, Line 45).

21. As per Claim 16, Kissell teaches: The method of claim 1, wherein:

the one or more shreds are to perform computation functions (Column 2, Line 34).

22. As per Claim 17, Kissell teaches: An apparatus, comprising:

execution resources to execute a first shared resource thread ("shred") concurrently with at least a second shred (Column 2, Line 36) in response to receiving a first user instruction (Column 10, Lines 37-45, FORK creates a new thread), but fails to teach:

a shared register to be directly accessible by a second user instruction from the first and the second shred to provide communication between the first shred and the second shred, wherein the shared register is not accessible shreds that do not share an application state with the first and the second shred.

Kissell teaches a shared memory area accessible by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with less storage available), and the system can work with either type, as memory is memory.

Furthermore, as discussed above, Kissell discloses that every process has memory shared between all threads. While Kissell does not elaborate on how this shared memory is used, Marshall, a reference which discloses well known concepts of threads and processes, discloses that the shared memory space of processes are used to share data between threads in a process (Page 2, where Marshall indicates that sharing data between threads is immediately available, while there is significant overhead in sharing data between processes). Therefore, one of ordinary skill in the art would have recognized, given how threads and processes work, that the shared memory space of the process, which is private to threads in the process, can be used to communicate data between threads, and as taught by Hennessy, could be implemented as registers if one wished to have higher speed communication at higher cost. Additionally, Examiner notes that in Kissell's description of the inter-thread communication storage, Kissell indicates that the storage "can" be made global, which implies that it is not necessarily inherently global. The inter-thread communication storage could easily be read as a description of the shared memory of a process which is made available to all threads of the system, as it describes portions of memory which threads can read or write from in order to pass data along to each other. Therefore, even without the well known teachings of Marshall and the private shared memory spaces in processes, Kissell could be interpreted as having the private storages for threads to communicate.

23. As per Claim 20, Kissell teaches the apparatus of claim 17, wherein the shared register comprises a first register to enable an operating system or BIOS to enable multithreading architecture extensions for user-level multithreading (Column 13, Lines 1-5).

24. As per Claim 22, Kissell teaches: The apparatus of claim 17, wherein the execution resources include one or more processor cores capable of executing multiple shreds concurrently (Column 2, Line 36).

25. As per Claim 23, Kissell teaches: The apparatus of claim 17, wherein the shared register is to hold at least a portion of a state shared among the first shred and the second shred (Figure 17 shows the shared state).

26. As per Claim 24, Kissell teaches: The apparatus of claim 17, wherein the first shred and the second shred share a current privilege level and share a common address translation (Column 9, Lines 41-43, threads in the same process share a common address space and can share the same privilege level).

27. As per Claim 25, Kissell teaches: The apparatus of claim 17, further comprising logic to execute a user-level instruction to create the first shred (Column 10, Lines 37-45, logic must inherently exist to support the instruction).

28. As per Claim 27, Kissell teaches: The apparatus of claim 17, further comprising a group of registers to share a system state among the first shred and the second shred (Columns 13, 14, and 15 all described other shared registers and the state they hold which is shared. Especially of note is A, D, G, and the Status register).

29. As per Claim 28, Kissell teaches: The apparatus of claim 17, further comprising a group of registers, which does not include the shared register, to hold a private portion of a state to be associated with the first shred (Column 7, Lines 53-56 and Column 9, Lines 43-45).

30. As per Claim 32, Kissell teaches: The apparatus of claim 17, further comprising:

a user-level exception mechanism to report an exception to the first shred (Column 9, Lines 61-67).

As per Claim 35, Kissell teaches: A non-transitory computer readable medium including data that, when accessed by a machine, cause the machine to perform operations comprising:

receiving user-level programming instructions to execute a plurality of user-level threads of execution (Column 10, Lines 37-45, FORK creates a new thread, and Column 8, Lines 40-42 show that the OS is not used), wherein each of the plurality of user-level threads of execution are to be associated with a private state of an operating system (OS)-generated thread (Column 9, Lines 43-45, each thread has a separate

state portion) and to share a system state with the OS-generated thread (Column 9, Lines 41-43, they share the same address space);

creating, in hardware without intervention of an OS, the plurality of user-level threads of execution to be executed concurrently on multiple instruction sequencers of a processor in the machine in response to receiving the user-level programming instruction (Column 10, Lines 37-45, Figure 3 shows multiple fetch/decode/execution pipelines, meaning multiple instruction sequencers), but fails to teach:

communicating between the plurality of user-level threads of execution via one or more shared registers that are to be directly updateable by the plurality of user-level threads, wherein the one or more shared registers are to not be accessible by shreds that do not share the system state with the OS-generated thread.

Kissell teaches a shared memory area updateable by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with

less storage available), and the system can work with either type, as memory is memory.

Furthermore, as discussed above, Kissell discloses that every process has memory shared between all threads. While Kissell does not elaborate on how this shared memory is used, Marshall, a reference which discloses well known concepts of threads and processes, discloses that the shared memory space of processes are used to share data between threads in a process (Page 2, where Marshall indicates that sharing data between threads is immediately available, while there is significant overhead in sharing data between processes). Therefore, one of ordinary skill in the art would have recognized, given how threads and processes work, that the shared memory space of the process, which is private to threads in the process, can be used to communicate data between threads, and as taught by Hennessy, could be implemented as registers if one wished to have higher speed communication at higher cost. Additionally, Examiner notes that in Kissell's description of the inter-thread communication storage, Kissell indicates that the storage "can" be made global, which implies that it is not necessarily inherently global. The inter-thread communication storage could easily be read as a description of the shared memory of a process which is made available to all threads of the system, as it describes portions of memory which threads can read or write from in order to pass data along to each other. Therefore, even without the well known teachings of Marshall and the private shared memory spaces in processes, Kissell could be interpreted as having the private storages for threads to communicate.

31. As per Claim 36, Kissell teaches: The computer readable medium of claim 35, wherein the operations further comprise:

the private state is associated with at least one of a plurality of registers including general purpose registers, floating point registers, MMX registers, segment registers, a flags register, an instruction pointer, control and status registers, SSE registers, and a MXCSR register (It is inherent that each thread of execution has these registers in their private state, or they cannot function as intended. Also see Column 9, Lines 43-45).

32. As per Claim 37, Kissell teaches: The computer readable medium of claim 35, wherein the private state includes a private portion of an application state (Column 9, Lines 43-45), and wherein the plurality of user-level threads of execution are also to share a shared portion of the application state (Column 9, Lines 41-43), the shared portion of the application state to be associated with at least one of a plurality of registers including a control register, a flags register, memory management registers, a local descriptor table register, a task register, debug registers, model specific registers, shared registers, and shred control registers (Column 13, Line 44, the ThreadControl register).

33. As per Claim 38, Kissell teaches: The computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

sharing a state among the plurality of user-level threads of execution (Figure 17 shows some examples of shared state; and

storing the state in one or more registers (Column 13, Line 44, the ThreadControl register).

34. As per Claim 39, Kissell teaches: The computer readable medium of claim 35, wherein the plurality of user-level threads of execution share a current privilege level and share a common address translation (Column 9, Lines 41-43, threads in the same process share a common address space and can share the same privilege level).

35. As per Claim 40, Kissell teaches: The computer readable medium of claim 35, wherein the user-level programming instructions include an instruction to create one or more of the plurality of user-level threads of execution (Column 10, Line 37).

36. As per Claim 41, Kissell teaches: The computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising communicating among the plurality of user-level threads of execution (Column 12, Lines 51-55).

37. As per Claim 42, Kissell teaches: The computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising sharing a system state among the plurality of user-

level threads of execution (Figure 17).

38. As per Claim 43, Kissell teaches: The article of computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising communicating between the plurality of user-level threads of execution via one or more shared registers (Column 12, Lines 57-60, the ThreadControl register).

39. As per Claim 44, Kissell teaches: The computer readable medium of claim 35, wherein an application program controls the plurality of user-level threads of execution directly (all instructions are part of a program, and the FORK instruction creates the threads, thus the program including the FORK controls the threads directly), including scheduling of the plurality of user-level threads of execution (Column 4, Lines 33-41), and wherein an operating system executed by the multiprocessor schedules the OS-generated thread (Kissell only teaches that user-instructions schedule and create the threads, yet teaches that processes exist (Column 8, Line 30). Therefore, the OS must be scheduling and creating the processes, since there is nothing else that could do it, and it is also well-established that OS's generally execute instructions and processes).

40. As per Claim 47, Kissell teaches the computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

reporting one or more exceptions to a first thread of the plurality of user-level threads of execution (Column 9, Lines 61-67).

41. As per Claim 50, Kissell teaches: The computer readable medium of claim 35, wherein the plurality of user-level threads of execution perform input/output (I/O) functions (Column 5, Line 45) and computation functions (Column 2, Line 34).

42. As per Claim 51, Kissell teaches: A system, comprising:

a microprocessor to implement an instruction set architecture (ISA) (Column 7, Lines 25-27), the microprocessor to execute multiple operating system (OS)-generated threads (Column 8, Line 30, Processes A and C), wherein the microprocessor is also to concurrently execute multiple shared resource threads (shreds) associated with an OS-generated thread (Column 4, Line 5), each of the multiple shreds to be associated with a private state within the OS-generated thread (Column 9, Lines 43-45) and a shared state of the OS-generated thread (Column 9, Lines 41-43);

wherein each of the threads is to be created utilizing hardware of the microprocessor in response to application program instructions of the ISA (Column 10, Lines 37-50); and

a memory to store one or more instructions of the ISA that is to allow user-level multithreading operations (Column 8, Lines 4-5), but fails to teach:

wherein a register that is part of the shared state of the OS-generated thread is adapted to be directly updateable by the multiple shreds to allow communication

between the multiple shreds, wherein the register is also adapted to not be accessible by shreds that are not associated with the shared state of the OS-generated thread.

Kissell teaches a shared memory area updateable by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with less storage available), and the system can work with either type, as memory is memory.

Furthermore, as discussed above, Kissell discloses that every process has memory shared between all threads. While Kissell does not elaborate on how this shared memory is used, Marshall, a reference which discloses well known concepts of threads and processes, discloses that the shared memory space of processes are used to share data between threads in a process (Page 2, where Marshall indicates that sharing data between threads is immediately available, while there is significant overhead in sharing data between processes). Therefore, one of ordinary skill in the art would have recognized, given how threads and processes work, that the shared

memory space of the process, which is private to threads in the process, can be used to communicate data between threads, and as taught by Hennessy, could be implemented as registers if one wished to have higher speed communication at higher cost. Additionally, Examiner notes that in Kissell's description of the inter-thread communication storage, Kissell indicates that the storage "can" be made global, which implies that it is not necessarily inherently global. The inter-thread communication storage could easily be read as a description of the shared memory of a process which is made available to all threads of the system, as it describes portions of memory which threads can read or write from in order to pass data along to each other. Therefore, even without the well known teachings of Marshall and the private shared memory spaces in processes, Kissell could be interpreted as having the private storages for threads to communicate.

43. As per Claim 55, Kissell teaches: A system, comprising:

a microprocessor (Column 7, Line 24); and

memory coupled to the microprocessor, the memory to store the one or more application program instructions (Column 8, Lines 4-5);

wherein the microprocessor is further to execute the plurality of user-level threads concurrently (Column 4, Line 5), but fails to teach:

including a plurality of user-level multithreading registers wherein the registers are addressable by one or more application program instructions from a plurality of user-level threads to support communication among the plurality of user-level threads.

Kissell teaches a shared memory area updateable by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with less storage available), and the system can work with either type, as memory is memory.

44. As per Claim 56, Kissell teaches: The system of claim 55, wherein the plurality of user-level multithreading registers further comprises a plurality of shared shred registers to facilitate communication between a plurality of shreds and to facilitate synchronization between the plurality of shreds (Column 12, Lines 50-60).

45. As per Claim 57, Kissell teaches: The system of claim 56, wherein the plurality of user-level multithreading registers further comprises a plurality of shred control registers to manage the plurality of shreds (Columns 13, 14, and 15, note the status register and ThreadControl registers in particular).

46. As per Claim 58, Kissell teaches The system of claim 57, wherein the microprocessor:

receives programming instructions to execute one or more shreds in accordance with the ISA (Column 10, Line 37);

configures one or more instruction sequencers via the ISA (Figure 3 shows the fetch and decode pipelines in which instruction sequencers exist); and

executes the one or more shreds concurrently (Column 4, Line 5 shows that it is a multithreaded processor, which means executing multiple threads concurrently).

47. As per Claim 62, Kissell teaches: The article of manufacture of claim 35, wherein the one or more user-level programming instructions include an instruction to destroy one or more of the plurality of shreds (Column 10, Line 51, YIELD).

48. As per Claim 63, Kissell teaches: The system of claim 51, wherein the one or more instructions include an instruction to create a shred without intervention of an operating system (Column 10, Line 37, FORK).

49. As per Claim 64, Kissell teaches: The system of claim 51, wherein the one or more instructions include an instruction to destroy a shred without intervention of an operating system (Column 10, Line 51, YIELD).

50. As per Claim 65, Kissell teaches: The system of claim 51, wherein: the user-level multi-threading operations include concurrent execution of two or more shreds associated with the same thread (Column 10, Lines 45-50, the new thread is part of the same process as the old one, as it is a duplicate at the time).

51. Claim 68 is rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell and Hennessy, further in view of Official Notice.

52. As per Claim 68, Kissell teaches: The processor of Claim 67, wherein:

the first group of resources, and the second group of resources include registers (Column 9, Lines 43-45 and Column 13, Line 44), and wherein the first user-level instruction and the second user-level instruction are an Instruction Set Architecture instructions to create a shred (Column 10, Line 37, the FORK instruction), which are to be associated with a first user software entity and a second user software entity, respectively (Column 8, Line 30), but fails to teach:

the third group of resources including registers.

Kissell teaches a shared memory area updateable by any thread to share and pass information between them (Column 16, Lines 37-53, the Inter-Thread Communication Storage), however, the shared memory area of Kissell is only disclosed as a general "memory", and not specifically as registers. However, as one of ordinary skill in the art would recognize, and as detailed by Hennessy, registers are simply a part of the memory hierarchy, where they are faster, but also most expensive than other

forms of memory (see Pages 40 and 41). Therefore, while Kissell may implement the shared storage in a cache memory (as an example), one of ordinary skill in the art would recognize that the same storage could be implemented in registers, with the benefit of having significantly higher speeds (but at a significantly increased cost or with less storage available), and the system can work with either type, as memory is memory.

53. Claims 13-14, 33-34, 48-49 and 59-61 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell, Marshall, and Hennessy, further in view of Golliver et al. (USPN 6,378,067, herein Golliver).

54. As per Claim 13, Kissell teaches the method of claim 1, but fails to teach:

handling with user-level exception handler code an exception generated during execution of the first shred, without intervention of the operating system.

Kissell teaches that exceptions are handled within his invention (see Column 9, Line 33 –Column 10, Line 22), however, Kissell does not teach that the exception is handled without intervention of the operating system. Kissell teaches that the thread itself invokes and executes the exception handler, but there is no explicit recitation that the OS is not called during the handler code's execution. Golliver teaches a method to handle multiple exceptions at once by prioritizing them, and dealing with them in a non-operating system exception handler (Column 3, Line 66 – Column 4, Line 6, and Column 4, Lines 22-25. The operating system is not used to process the exception

unless explicitly called, as shown in Column 7, Lines 22-27). Given the need for a method to handle exceptions, and the desire to minimize the operating systems involvement in thread processing in Kissell's invention, and additionally the need to handle multiple exceptions which could come up when executing multiple shreds concurrently, it would have been obvious to one of ordinary skill in the art at the time the invention was made to use Golliver's exception handler to handle Kissell's exceptions.

55. As per Claim 14, Golliver teaches: The method of claim 13, further comprising:

receiving the exception from an application program (Column 4, Lines 22-26); and

determining whether to report the exception to the operating system (Column 7, Lines 25-27).

56. As per Claim 33, Kissell teaches the apparatus of claim 17, but fails to teach:

an exception mechanism to report an exception to an operating system.

Kissell teaches that exceptions are handled within his invention (see Column 9, Line 33 -–Column 10, Line 22), however, Kissell does not teach that the exception is handled using the operating system. Kissell teaches that the thread itself invokes and executes the exception handler, but there is no explicit recitation that the OS is called during the handler code's execution. Golliver teaches a method to handle multiple exceptions at once by prioritizing them, and dealing with them in a non-operating system exception handler (Column 3, Line 66 – Column 4, Line 6, and Column 4, Lines

22-25. The operating system is not used to process the exception unless explicitly called, as shown in Column 7, Lines 22-27). Given the need for a method to handle exceptions, and the desire to minimize the operating systems involvement in thread processing in Kissell's invention, and additionally the need to handle multiple exceptions which could come up when executing multiple shreds concurrently, it would have been obvious to one of ordinary skill in the art at the time the invention was made to use Golliver's exception handler to handle Kissell's exceptions.

57. As per Claim 34, Golliver teaches: The apparatus of claim 32, further comprising:

a mechanism to a first exception associated with the first shred and a second exception associated with the second shred (Column 3, Line 66 – Column 4, Line 6, and Column 5, Lines 29-31);

wherein the exception mechanism includes a prioritizer to prioritize the first and the second exceptions (Column 4, Line 1); and

wherein the exception mechanism is further to report only one of the prioritized first and second exceptions at a time to the operating system (Column 7, Lines 23-28 disclose the operating system may be able to be told to handle exceptions, and Column 7, Lines 11-13 show that the faults can be reported one at a time).

58. As per Claim 48, Kissell teaches: The computer readable medium of claim 47, but fails to teach:

wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

reporting the one or more exceptions from an application program; and

determining whether to report the one or more exceptions to an operating system.

Kissell teaches that exceptions are handled within his invention (see Column 9, Line 33 –Column 10, Line 22), however, Kissell does not teach that the exception is handled using the operating system. Kissell teaches that the thread itself invokes and executes the exception handler, but there is no explicit recitation that the OS is called during the handler code's execution. Golliver teaches a method to handle multiple exceptions at once by prioritizing them, and dealing with them in a non-operating system exception handler (Column 3, Line 66 – Column 4, Line 6, and Column 4, Lines 22-25. The operating system is not used to process the exception unless explicitly called, as shown in Column 7, Lines 22-27). Given the need for a method to handle exceptions, and the desire to minimize the operating systems involvement in thread processing in Kissell's invention, and additionally the need to handle multiple exceptions which could come up when executing multiple shreds concurrently, it would have been obvious to one of ordinary skill in the art at the time the invention was made to use Golliver's exception handler to handle Kissell's exceptions.

59. As per Claim 49, Golliver teaches: The computer readable medium of claim 48, wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

prioritized-reporting of the one or more exceptions to the operating system; comprising receiving the one or more exceptions concurrently via different user-level threads of the plurality of user-level threads of execution (Column 5, Lines 30-32); and

servicing one of the one or more exceptions according to the prioritized-reporting while suspending exception processing of other exceptions of the one or more exceptions (Column 7, Lines 11-13).

60. As per Claim 59, Golliver teaches: The apparatus of claim 32, wherein: the user-level exception mechanism is further to vector to a fixed location in order to allow the shred to service to the exception (Figure 4, and Column 7, Lines 23-28, the user handler set).

61. As per Claim 60, Golliver teaches: The apparatus of claim 32, wherein:

the plurality of instructions further include a system call instruction to explicitly invoke an operating system to service to the exception (Column 7, Lines 23-28).

62. As per Claim 61, Golliver teaches: The apparatus of claim 34, wherein said prioritizer employs a round-robin scheme (Column 7, Lines 11-12 and Column 5, Lines 30-32).

63. Claims 21 and 66 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell, Marshall, and Hennessy, in view of Official Notice.

64. As per Claim 21, Kissell teaches: The apparatus of claim 17, but fails to teach:

wherein the shared register comprises a first register to be atomically updated to synchronize data between the first and the second shred.

While Kissell does not teach that the shared registers are updated atomically, Examiner is taking Official Notice that it is well known in the art that if a shared resource is being updated, it must be done atomically to prevent a resource conflict which would cause the system to execute incorrectly (with perhaps Thread A writing data for Thread B, but Thread B would read it too early if Thread A does not write atomically). Given this, one of ordinary skill in the art would have recognized that writing to a shared register requires atomicity to ensure data coherency.

65. As per Claim 66, Kissell teaches: The system of Claim 55, but fails to teach:

wherein the memory is from a plurality of memory devices including DRAM, flash, and EEPROM.

While Kissell does not explicitly teach DRAM, flash, or EEPROM, Examiner is taking Official Notice that these types of memory are extremely well known in the art, and one of ordinary skill in the art would have made use of them in Kissell depending upon their implementation.

66. Claim 26 is rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell, Marshall, and Patterson, further in view of Wendorf et al. (USPN 5,845,129, herein Wendorf).

67. As per Claim 26, Kissell teaches: The apparatus of claim 17, but fails to teach:

wherein the shared register comprises a first register to be utilized as a register semaphore to synchronize data between the first and the second shred.

While Kissell does not teach of a semaphore, Examiner notes that one of ordinary skill in the art would recognize that in order to update shared values and maintain data coherence, atomicity must be used to ensure that nothing reads from the location being updated before the update is finished. One of ordinary skill in the art would also clearly recognize that a semaphore is a well-known method of implementing this. Additionally, Examiner notes Wendorf, entered as extrinsic evidence, which discloses that a common thread-to-thread synchronization tool (Column 7, Lines 30-37). Given this, one of ordinary skill in the art would have been motivated to provide a semaphore in order to maintain data coherency and ensure correct execution of a program.

68. Claims 45-46 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell, Marshall, and Hennessy, in view of Foldoc.

69. As per Claim 45, Kissell teaches: The computer readable medium of claim 35, wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

associating the plurality of user-level threads of execution with the OS-generated thread (Column 9, Lines 41-43, each thread is associated with a processor (OS thread)), but fails to teach:

suspending the plurality of user-level threads of execution belonging to the OS-generated thread when a context switch request is received through a single one of the plurality of threads of execution.

Kissell teaches that there is context switching, however, does not teach that all the threads associated with a process would be suspended during a context switch . However, Foldoc (Context switch) teaches that on a context switch between processes, the entire process stops running, and another begins. Given that the reason to multithread is to maximize the shared state as much as possible, it would have been obvious to one of ordinary skill in the art at the time the invention was made to shut down all the shreds associated with the thread/process when it was switched out, so the new process could make use of the advantages of parallelism without the other shreds getting in the way.

70. As per Claim 46, Kissell teaches: The computer readable medium of claim 45, wherein the computer readable medium further includes data that causes the machine to perform operations comprising:

storing one or more threads of execution states associated with the plurality of user-level threads of execution when the context switch request is received (inherent that this occurs in context switching).

71. Claim 69 is rejected under 35 U.S.C. 103(a) as being unpatentable over Kissell and Hennessy, in view of Marr et al. (United States Patent Application Publication 2003/0126416, herein Marr).

72. As per Claim 69, Kissell teaches: The apparatus of Claim 68, but fails to teach:

further comprising a hardware re-namer to allocate the first group of registers as private registers and the third group of registers as shared registers based on a bit vector.

Kissell does not teach allocating registers as either private or shared based on a bit vector. However, Marr teaches a multithreaded system similar to Kissell, wherein processor throughput can be increased by allowing threads to access registers from other threads (thus making a private register shared) when certain conditions arise (Paragraphs 24 and 31). Additionally, Marr has some embodiments where registers (among other resources) can be designated as either shared, duplicated (private), or partitioned (also private), see Paragraph 33. Given that each register would need to know what can access it, a bit vector would almost necessarily be required, and if not, one of ordinary skill in the art would recognize that a bit vector would be one of the simplest implementations in order to carry out Marr's invention. Given the advantages

presented in Marr, one of ordinary skill in the art would have been motivated to combine the teachings of Kissell and Marr to allow registers to be allocated as private and shared.

Response to Arguments

73. Examiner notes that the 101 rejections of the claims have been overcome due to Applicant's amendment.

74. Examiner notes that in the previous action, Examiner made note of the rejection to Claim 57 remaining unchanged, Examiner intended to refer to Claim 67. Additionally, in the independent claims, when referring to the inter-thread communication storage, Examiner referred to Column 17, when Examiner intended to refer to Column 16. Examiner has corrected the typographical errors in this action, and apologizes for any confusion caused.

75. In regard to Applicant's arguments regarding Claims 67 and 70-71, Applicant has argued that Kissell teaches inter-thread communication for synchronizing threads that are executing on different VPE's, and not for private communication between shreds of an OS thread. However, Examiner does not agree with the Applicant's interpretation, and even if we assume that is all the inter-thread communication storage can do, Examiner does not see how data for synchronizing a thread is not a "shared application state". There does not appear to be any explicit recitation of what the shared application

state in Claim 67 is supposed to represent, therefore, if two threads are "synchronizing" to each other, then they would appear to be sharing some kind of state (and the recitation of the "synchronizing" data, as disclosed by Kissell, appears to be shared or passed data, as it refers to data stored by one thread and loaded by another).

76. Applicant has further argued, in regard to the other independent claims, that Kissell and Hennessy, in combination, do not teach "wherein the one or more shared registers are private to shreds that share the shared portion of the first application state and the first system state". Applicant has argued that the inter-thread communication memory in Kissell enables synchronization between all threads on a processor through use of global access to the physical address space, and therefore there is no disclosure in Kissell that the inter-shred communication memory is to be private amongst shreds sharing a certain state. However, Examiner notes that in Kissell's disclosure of the inter-thread communication storage, in Column 9, Lines 17-20, that he discloses that it "can" be made global, which implies that it is not necessarily global, and may in fact just be referring to embodiments where the shared memory space of the process is used to allow threads in the process to communicate with each other as is well known in the art.

However, Examiner has further provided a reference, Marshall, which teaches well known concepts of threads and processes as are known in the art. Marshall further elaborates on the shared memory space of processes, and notes that it allows threads inside a process to communicate with each other and share data, because they share the same address space. Marshall differentiates this from communication between

processes, which has significantly higher overhead to communicate, because they do not share address space (because it is private). Therefore, one of ordinary skill in the art would have recognized that the shared memory space of Kissell would be used for threads sharing the system and application state to communicate and share data with one another, and that said memory space would be private to threads outside that application state. And given the teachings of Hennessy, one of ordinary skill in the art would recognize that the memory could be embodied as registers if the need for speed outweighed the higher cost and lower storage space compared to other types of memory. For these reasons, Examiner asserts that the claims are still obvious in view of the prior art, as either the inter-thread communication storage could read on the claims, or alternatively the shared memory space of the process could read on the limitation.

Conclusion

77. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL.** See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any

extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert Fennema whose telephone number is (571)272-2748. The examiner can normally be reached on Monday-Thursday, 9:30-6:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Robert Fennema/
Examiner
Art Unit 2183